**Search Results** - Record(s) 1 through 14 of 14 returned.

☐ 1. Document ID: US 6629123 B1

L24: Entry 1 of 14 File: USPT Sep 30, 2003

US-PAT-NO: 6629123
DOCUMENT-IDENTIFIER: US 6629123 B1

TITLE: Interception of unit creation requests by an automatic distributed partitioning system

DATE-ISSUED: September 30, 2003

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Hunt; Galen C. | Bellevue | WA | | |

US-CL-CURRENT: 709/106; 709/310, 717/131

ABSTRACT:

An automatic distributed partitioning system (ADPS) intercepts function calls to unit activation functions that dynamically create application units, such as a component instantiation function. A system service library provides a unit activation function. An application program includes at least one function call to the unit activation function. The ADPS redirects the function call to instrumentation of the ADPS. In one technique, the ADPS uses inline redirection of the function call to the unit activation function.

23 Claims, 18 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 18

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw Desc | Image

☐ 2. Document ID: US 6381735 B1

L24: Entry 2 of 14 File: USPT Apr 30, 2002

US-PAT-NO: 6381735
DOCUMENT-IDENTIFIER: US 6381735 B1

TITLE: Dynamic classification of sections of software

DATE-ISSUED: April 30, 2002

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Hunt; Galen C. | Bellevue | WA | | |

US-CL-CURRENT: 717/158

ABSTRACT:

Dynamic classification of sections of software using a profile-based optimization system optimizes management of the sections of software. Software executes under expected usage conditions. After execution, a set of usage profiles describes the dynamic properties of sections of the software. Each usage profile includes information identifying a section of software. Each usage profile maps to an outcome meant to optimize management of the sections of the software during later execution. During such later execution, a usage background describes the dynamic properties of a section of the software. The usage background includes information identifying the section of software. By matching the usage background to a usage profile in the set of usage profiles, the section is dynamically classified during later execution. Based on this dynamic classification, the section maps to the outcome meant to optimize management of the sections of software.

61 Claims, 18 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 18



---

☐ 3. Document ID: US 6282701 B1

L24: Entry 3 of 14 File: USPT Aug 28, 2001

US-PAT-NO: 6282701
DOCUMENT-IDENTIFIER: US 6282701 B1

TITLE: System and method for monitoring and analyzing the execution of computer programs

DATE-ISSUED: August 28, 2001

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Wygodny; Shlomo | Ramat Hasharon | | | IL |
| Barboy; Dmitry | Rehovot | | | IL |
| Prouss; Georgi | Kiev | | | UA |
| Vorobey; Anatoly | Rishon Lezion | | | IL |

US-CL-CURRENT: 717/125; 702/183, 702/187, 709/224, 709/317, 709/331, 714/35, 714/38, 714/45, 714/46, 717/128, 717/163

ABSTRACT:

A software system is disclosed which facilitates the process of tracing the execution paths of a program, called the client. The tracing is performed without requiring modifications to the executable or source code files of the client. Trace data collected during the tracing operation is collected according to instructions in a trace options file. At run time, the tracing library attaches to the memory image of the client. The tracing library is configured to monitor execution of the client and to collect trace data, based on selections in the trace options file. The developer then uses a trace analyzer program, also having a graphical user interface, to view the trace information. The system can trace multiple threads and multiple processes. The tracing library is preferably configured to runs in the same process memory space as the client thereby tracing the execution of the client program without the need for context switches. The tracing system provides a remote mode and an online mode. In remote mode, the developer sends the trace control information to a remote user site together with a small executable image called the agent that enables a remote customer, to generate a trace file that represents execution of the client application at the remote site. In online mode, the developer can generate trace options, run and trace the client, and display the trace results in near real-time on the display screen during execution of the client

program.

31 Claims, 17 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 15

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments  KWIC | Draw Desc | Image

---

☐ 4. Document ID: US 6263491 B1

L24: Entry 4 of 14 File: USPT Jul 17, 2001

US-PAT-NO: 6263491
DOCUMENT-IDENTIFIER: US 6263491 B1

TITLE: Heavyweight and lightweight instrumentation

DATE-ISSUED: July 17, 2001

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Hunt; Galen C. | Bellevue | WA | | |

US-CL-CURRENT: 717/130

ABSTRACT:

An instrumentation system performs operations such as profiling an application and partitioning and distributing units of the application using different versions of metadata describing the application. Performing an operation on an executing application generates overhead. Detailed metadata used in operations such as profiling create unnecessary overhead during other operations. By removing metadata detail unnecessary for a particular operation, an instrumentation system using reduced metadata generates less overhead for that particular operation. Different instrumentation packages include different versions of metadata for performing operations on the application.

36 Claims, 18 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 18

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments  KWIC | Draw Desc | Image

---

☐ 5. Document ID: US 6260150 B1

L24: Entry 5 of 14 File: USPT Jul 10, 2001

US-PAT-NO: 6260150
DOCUMENT-IDENTIFIER: US 6260150 B1

TITLE: Foreground and background context controller setting processor to power saving mode when all contexts are inactive

DATE-ISSUED: July 10, 2001

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Diepstraten; Wilhelmus J. M. | Haghorst | | | NL |
| Fischer; Michael A. | San Antonio | TX | | |
| Hardell; Wesley D. | San Antonio | TX | | |

US-CL-CURRENT: 713/323; 709/107, 712/229

ABSTRACT:

A context controller for managing multitasking in a processor and a method of operating the same. In one embodiment, the context controller includes: (1) foreground and background task controllers that allocate processor resources to active contexts corresponding to foreground and background tasks, respectively, and (2) mode switching circuitry, coupled to the foreground and background task controllers, that places the processor in an idle state and a power saving mode when all of the contexts are inactive.

22 Claims, 21 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 21



---

☐ 6. Document ID: US 6243736 B1

L24: Entry 6 of 14 File: USPT Jun 5, 2001

US-PAT-NO: 6243736
DOCUMENT-IDENTIFIER: US 6243736 B1

TITLE: Context controller having status-based background functional task resource allocation capability and processor employing the same

DATE-ISSUED: June 5, 2001

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Diepstraten; Wilhelmus J. M. | Haghorst | | | NL |
| Fischer; Michael A. | San Antonio | TX | | |
| Hardell; Wesley D. | San Antonio | TX | | |

US-CL-CURRENT: 709/108; 709/103, 709/312

ABSTRACT:

A context controller for managing multitasking in a processor and a method of operating the same. In one embodiment, the context controller includes: (1) memory that contains contexts corresponding to background tasks to be executed in the processor, the contexts having status indicators associated therewith and (2) a background task controller that reads the status indicators associated with the contexts and cyclicly activates the contexts based on the status indicators.

22 Claims, 21 Drawing figures
Exemplary Claim Number: 1,15
Number of Drawing Sheets: 21



---

☐ 7. Document ID: US 6205468 B1

L24: Entry 7 of 14 File: USPT Mar 20, 2001

US-PAT-NO: 6205468
DOCUMENT-IDENTIFIER: US 6205468 B1

TITLE: System for multitasking management employing context controller having event vector selection by priority encoding of contex events

DATE-ISSUED: March 20, 2001

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Diepstraten; Wilhelmus J. M. | Haghorst | | | NL |
| Fischer; Michael A. | San Antonio | TX | | |
| Hardell; Wesley D. | San Antonio | TX | | |

US-CL-CURRENT: 709/108; 709/100, 709/101, 709/102, 709/103, 709/104, 709/105, 709/106, 709/107, 712/228, 712/229

ABSTRACT:

A context controller for managing multitasking in a processor and a method of operating the same. In one embodiment, the context controller includes: (1) an event recorder that records occurrences of events and (2) an encoder, associated with the event recorder, that, in response to a software instruction, priority encodes bits corresponding to at least some of the events to generate therefrom an event-dependent vector to allow the processor to branch as a function thereof. Vectoring is per-instance of the vector decode software instruction, not per-event or per-context.

22 Claims, 21 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 21

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

---

☐ 8. Document ID: US 6202199 B1

L24: Entry 8 of 14 File: USPT Mar 13, 2001

US-PAT-NO: 6202199
DOCUMENT-IDENTIFIER: US 6202199 B1

TITLE: System and method for remotely analyzing the execution of computer programs

DATE-ISSUED: March 13, 2001

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Wygodny; Shlomo | Ramat Hasharon | | | IL |
| Barboy; Dmitry | Rehovot | | | IL |
| Prouss; Georgi | Kiev | | | UA |
| Vorobey; Anatoly | Rishon Lezion | | | IL |

US-CL-CURRENT: 717/125; 702/183, 717/128

ABSTRACT:

A software system is disclosed which facilitates the process of tracing the

execution paths of a program, called the client. The tracing is performed without requiring modifications to the executable or source code files of the client. Trace data collected during the tracing operation is collected according to instructions in a trace options file. At run time, the tracing library attaches to the memory image of the client. The tracing library is configured to monitor execution of the client and to collect trace data, based on selections in the trace options file. The developer then uses a trace analyzer program, also having a graphical user interface, to view the trace information. The system can trace multiple threads and multiple processes. The tracing library is preferably configured to runs in the same process memory space as the client thereby tracing the execution of the client program without the need for context switches. The tracing system provides a remote mode and an online mode. In remote mode, the developer sends the trace control information to a remote user site together with a small executable image called the agent that enables a remote customer, to generate a trace file that represents execution of the client application at the remote site. In online mode, the developer can generate trace options, run and trace the client, and display the trace results in near real-time on the display screen during execution of the client program.

33 Claims, 17 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 15

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments  RWIC | Draw Desc | Image

---

☐ 9. Document ID: US 6058465 A

L24: Entry 9 of 14 File: USPT May 2, 2000

US-PAT-NO: 6058465
DOCUMENT-IDENTIFIER: US 6058465 A

TITLE: Single-instruction-multiple-data processing in a multimedia signal processor

DATE-ISSUED: May 2, 2000

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Nguyen; Le Trong | Monte Sereno | CA | 95030 | |

US-CL-CURRENT: 712/7; 708/441, 712/13, 712/228, 712/23, 712/4, 712/5

ABSTRACT:

A vector processor architecture provides vector registers of fixed size having data elements of programmable size and type. The type and size for data elements are defined by instructions which manipulate operands associated with the vector registers. The data size defined by an instruction determines the number of the data elements in a vector register and the number of parallel operations performed to complete the instruction. One embodiment of the invention supports 8-bit, 9-bit, 16-bit, and 32-bit data element sizes of integer type for all sizes and floating point data type for the 32-bit data elements.

14 Claims, 12 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 10

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments  RWIC | Draw Desc | Image

---

☐ 10. Document ID: US 6052773 A

L24: Entry 10 of 14 File: USPT Apr 18, 2000

US-PAT-NO: 6052773
DOCUMENT-IDENTIFIER: US 6052773 A

TITLE: DPGA-coupled microprocessors

DATE-ISSUED: April 18, 2000

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| DeHon; Andre | Cambridge | MA | | |
| Bolotski; Michael | Somerville | MA | | |
| Knight, Jr.; Thomas F. | Belmont | MA | | |

US-CL-CURRENT: 712/43; 712/229

ABSTRACT:

A single chip microprocessor or memory device has reprogrammable characteristics according to the invention. In the case of the microprocessor, a fixed processing cell is provided as is common to perform logic calculations. A portion of the chip silicon real-estate, however, is dedicated a programmable gate array. This feature enables application-specific configurations to allow adaptation to the particular time-changing demands of the microprocessor and provide the functionality required to best serve those demands. This yields application acceleration and in system-specific functions. In other cases the configurable logic acts as network interface, which allows the same basic processor design to function in any environment to which the interface can adapt.

The invention also concerns a memory device having a plurality of memory banks and configurable logic units associated with the memory banks. An interconnect is provided to enable communication between the configurable logic units. These features lessen the impact of the data bottle-neck associated with bus communications, since the processing capability is moved to the memory in the form programmable logic, which can be configured to the needs of the specific application. The inherently large on-chip bandwidth can then be utilized to increase the speed at which bulk data is processed.

22 Claims, 34 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 30



---

☐ 11. Document ID: US 5953530 A

L24: Entry 11 of 14 File: USPT Sep 14, 1999

US-PAT-NO: 5953530
DOCUMENT-IDENTIFIER: US 5953530 A
**** See image for Certificate of Correction ****

TITLE: Method and apparatus for run-time memory access checking and memory leak detection of a multi-threaded program

DATE-ISSUED: September 14, 1999

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Rishi; Alok | El Granada | CA | | |
| Masamitsu; Jon A. | Livermore | CA | | |

US-CL-CURRENT: 717/127; 714/38, 714/48

ABSTRACT:

The present invention is a system and method for a "debugger Run-Time-Checking for valid memory accesses for multi-threaded application programs" (hereinafter "RTC/MT") wherein a run-time process which includes multiple threads running either serially or concurrently, may be monitored by a debugger program and memory access errors detected and correctly attributed to the process thread encountering the error. The RTC/MT system of the present invention also provides an apparatus and method which monitors and reports memory leaks as required for multi-threaded target programs.

25 Claims, 8 Drawing figures
Exemplary Claim Number: 18
Number of Drawing Sheets: 8

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments  KWIC | Draw Desc | Image

---

☐ 12. Document ID: US 5946474 A

L24: Entry 12 of 14 File: USPT Aug 31, 1999

US-PAT-NO: 5946474
DOCUMENT-IDENTIFIER: US 5946474 A

TITLE: Simulation of computer-based telecommunications system

DATE-ISSUED: August 31, 1999

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Skogby; Staffan | Danderyd | | | SE |

US-CL-CURRENT: 703/13; 709/100, 709/107, 709/321, 714/25, 714/30

ABSTRACT:

A simulation system (200) executes on a host computer system (110) for simulating a target telecommunications system. The simulation system (200) includes a simulation kernel (231) which contains a plurality of simulation subsystems, each of the simulation subsystems corresponding to subsystems of the target telecommunications system. The simulation kernel (231) is assembled as one executable image with all simulation subsystems executing in one common process context and sharing one common execution thread. The simulation system includes a central processor simulator (234) which simulates execution of target instructions in response to commands entered through a user communication channel to the simulation kernel. The central processor simulator simulates utilization of registers of the central processor of the target telecommunication system. An internal bus of the target telecommunication system is simulated by function calls between the simulation subsystems.

19 Claims, 41 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 31

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments  KWIC | Draw Desc | Image

☐ 13. Document ID: US 5889988 A

L24: Entry 13 of 14 File: USPT Mar 30, 1999

US-PAT-NO: 5889988
DOCUMENT-IDENTIFIER: US 5889988 A

TITLE: Debugger for debugging tasks in an operating system virtual device driver

DATE-ISSUED: March 30, 1999

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Held; James P. | Portland | OR | | |

US-CL-CURRENT: 709/103; 709/107

ABSTRACT:

A debugger that is multi-task aware and capable of providing symbolic support to a graphical user interface (GUI) is disclosed. The debugger disclosed communicates with a multi-tasking kernel nested within a driver of the operating system within the 0 privilege level. The multi-tasking kernel distinguishes among the rest of its environment where a graphical user interface executes the driver tasks being debugged. The multi-tasking kernel, in cooperation with the debugger runs each element on a different thread of the same machine, thereby allowing the debugger and the driver tasks being debugged to continue to run without stopping operation of either the graphical user interface or the operating system associated with the graphical user interface.

19 Claims, 8 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 8

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | KWIC | Draw Desc | Image

☐ 14. Document ID: US 5179702 A

L24: Entry 14 of 14 File: USPT Jan 12, 1993

US-PAT-NO: 5179702
DOCUMENT-IDENTIFIER: US 5179702 A

TITLE: System and method for controlling a highly parallel multiprocessor using an anarchy based scheduler for parallel execution thread scheduling

DATE-ISSUED: January 12, 1993

INVENTOR-INFORMATION:

| NAME | CITY | STATE | ZIP CODE | COUNTRY |
|---|---|---|---|---|
| Spix; George A. | Eau Claire | WI | | |
| Wengelski; Diane M. | Eau Claire | WI | | |
| Hawkinson; Stuart W. | Eau Claire | WI | | |
| Johnson; Mark D. | Eau Claire | WI | | |
| Burke; Jeremiah D. | Eau Claire | WI | | |
| Thompson; Keith J. | Eau Claire | WI | | |
| Gaertner; Gregory G. | Eau Claire | WI | | |
| Brussino; Giacomo G. | Eau Claire | WI | | |
| Hessel; Richard E. | Altoona | WI | | |
| Barkai; David M. | Eau Claire | WI | | |
| Chen; Steve S. | Chippewa Falls | WI | | |
| Oslon; Steven G. | Chippewa Falls | WI | | |
| Strout, II; Robert E. | Livermore | CA | | |
| Masamitsu; Jon A. | Livermore | CA | | |
| Cox; David M. | Livermore | CA | | |
| O'Gara; Linda J. | Livermore | CA | | |
| O'Hair; Kelly T. | Livermore | CA | | |
| Seberger; David A. | Livermore | CA | | |
| Rasbold; James C. | Livermore | CA | | |
| Cramer; Timothy J. | Pleasanton | CA | | |
| Van Dyke; Don A. | Pleasanton | CA | | |
| Chandramouli; Ashok | Fremont | CA | | |

US-CL-CURRENT: 709/102; 709/104, 709/106, 717/124, 717/146, 717/151

ABSTRACT:

An integrated software architecture for a highly parallel multiprocessor system having multiple tightly-coupled processors that share a common memory efficiently controls the interface with and execution of programs on such a multiprocessor system. The software architecture combines a symmetrically integrated multithreaded operating system and an integrated parallel user environment. The operating system distributively implements an anarchy-based scheduling model for the scheduling of processes and resources by allowing each processor to access a single image of the operating system stored in the common memory that operates on a common set of operating system shared resources. The user environment provides a common visual representation for a plurality of program development tools that provide compilation, execution and debugging capabilities for multithreaded user programs and assumes parallelism as the standard mode of operation.

12 Claims, 60 Drawing figures
Exemplary Claim Number: 1
Number of Drawing Sheets: 53

# WEST Search History

DATE: Wednesday, October 22, 2003

| Set Name side by side | Query | Hit Count | Set Name result set |
|---|---|---|---|
| | *DB=USPT; PLUR=YES; OP=ADJ* | | |
| L24 | L23 and (context near (switch or swap)) | 14 | L24 |
| L23 | L22 | 65 | L23 |
| | *DB=USPT,PGPB; PLUR=YES; OP=ADJ* | | |
| L22 | L21 and breakpoint | 98 | L22 |
| L21 | L20 and debug$ | 270 | L21 |
| L20 | L18 and (thread or multithread$ or multi?thread$) | 758 | L20 |
| L19 | L18 and l1 | 13 | L19 |
| L18 | L17 or l16 or l15 | 2718 | L18 |
| L17 | ((712/228 \|712/229 )!.CCLS. ) | 590 | L17 |
| L16 | ((709/106 \|709/107 \|709/108 )!.CCLS. ) | 973 | L16 |
| L15 | ((717/124 \|717/125 \|717/126 \|717/127 \|717/128 \|717/129 \|717/130 \|717/131 \|717/132 \|717/133 \|717/158 )!.CCLS. ) | 1256 | L15 |
| | *DB=TDBD; PLUR=YES; OP=ADJ* | | |
| L14 | context swap | 5 | L14 |
| | *DB=USPT; PLUR=YES; OP=ADJ* | | |
| L13 | L12 | 4 | L13 |
| | *DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ* | | |
| L12 | multithread$ and context swap | 13 | L12 |
| L11 | L10 and debug$ | 42 | L11 |
| L10 | multithread$ and context switches | 166 | L10 |
| L9 | multithreading and context switches | 90 | L9 |
| | *DB=USPT; PLUR=YES; OP=ADJ* | | |
| L8 | L7 and thread | 23 | L8 |
| L7 | (vo, ted).xa. | 112 | L7 |
| | *DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ* | | |
| L6 | L5 and debug$ | 45 | L6 |
| L5 | L4 and swap | 140 | L5 |
| L4 | (thread or multithread) near2 processor | 1430 | L4 |
| L3 | L2 and debug$ | 6 | L3 |
| L2 | L1 and (thread or multithread) | 59 | L2 |
| L1 | micro-engine or microengine | 348 | L1 |

END OF SEARCH HISTORY

# WEST Search History

DATE: Monday, October 20, 2003

| Set Name side by side | Query | Hit Count | Set Name result set |
|---|---|---|---|
| *DB=TDBD; PLUR=YES; OP=ADJ* | | | |
| L14 | context swap | 5 | L14 |
| *DB=USPT; PLUR=YES; OP=ADJ* | | | |
| L13 | L12 | 4 | L13 |
| *DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ* | | | |
| L12 | multithread$ and context swap | 13 | L12 |
| L11 | L10 and debug$ | 42 | L11 |
| L10 | multithread$ and context switches | 166 | L10 |
| L9 | multithreading and context switches | 90 | L9 |
| *DB=USPT; PLUR=YES; OP=ADJ* | | | |
| L8 | L7 and thread | 23 | L8 |
| L7 | (vo, ted).xa. | 111 | L7 |
| *DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ* | | | |
| L6 | L5 and debug$ | 45 | L6 |
| L5 | L4 and swap | 140 | L5 |
| L4 | (thread or multithread) near2 processor | 1430 | L4 |
| L3 | L2 and debug$ | 6 | L3 |
| L2 | L1 and (thread or multithread) | 59 | L2 |
| L1 | micro-engine or microengine | 347 | L1 |

END OF SEARCH HISTORY

L14: Entry 1 of 5 File: TDBD Nov 1, 1993

TDB-ACC-NO: NN9311595

DISCLOSURE TITLE: Enhanced Implementation of Compare-and-Swap

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, November 1993, US

VOLUME NUMBER: 36
ISSUE NUMBER: 11
PAGE NUMBER: 595 - 598

PUBLICATION-DATE: November 1, 1993 (19931101)

CROSS REFERENCE: 0018-8689-36-11-595

DISCLOSURE TEXT:

The original implementation of Compare-and-Swap has an anomaly that can be eliminated by means of an enhancement described here. The enhancement is an alternate implementation of load with linking and conditional stores that have been implemented recently on some commercial multiprocessors. - The proposal by 1ù is subject to thrashing among multiprocessors. The present proposal reduces the thrashing somewhat by delaying exclusive access to a variable until a store actually has to be done. It also protects against incorrect behavior in a uniprocessor multiprogramming environment by assuring that the variable reserved is correctly associated with the conditional store. Compare-and-Swap is a Read/Modify/Write instruction that is used for synchronizing processors and processes 1ù. However, its use is subject to a subtle failure mode. The failure mode occurs in single-operand implementations of the instruction, but can be remedied in practice by using a double-operand implementation. Even the double-operand version is subject to rare failures, although the probability of such a failure is essentially zero. - An improved version of Compare-and-Swap as described in 2ù, essentially corrected the problem. This improvement eliminates a potential for blocking indefinitely long. It also is an enhancement to the software technique described by 2ù for dealing with the failure mode of Compare-and-Swap with a double-operand version of the instruction. - The enhancement provides additional protection for the Compare-and-Swap against the anomalous behavior. Commercial machines have evolved independently along a different tack, and tend to use the idea of the reservation as put forth in 1ù, coupled with a conditional store. Such implementations are available on the MIPS R-4000, DEC Alpha, and Sparc 9 microprocessors, and they are somewhat different from the implementation described here. They

- provide for a single active reservation per processor. This implementation provides for many active reservations per processor, and thereby can eliminate some potential thrashing among processes on a single processor. Compare-and-Swap works as follows when a processor is required to update a shared variable. 1. The processor obtains a local copy of the shared variable. 2. The processor locally computes a new value of the variable. 3. The processor issues a Compare-and-Swap instruction with the following information: a. The address in memory of the shared variable. b. The original value of the variable that was used locally. c. The updated value to be stored in memory. 4. If the old value agrees with the current value, then the assumption is that no other processor has touched the variable since it was obtained by the present processor. In this case, the new value is stored in memory. - If the old value disagrees with the current value, then some other processor has modified the variable. In this case, the current value is retrieved and replaces the original local copy. - The instruction returns a condition code that indicates which of the two possibilities occurred. 5. If the Compare-and-Swap failed, the program repeats the update computation using the value that has just been retrieved as the initial value of the shared variable. - The failure mode is often called the A-B-A Problem in that it occurs when an operand takes on the values A, then B, then A between accesses by a Compare-and-Swap. The Compare-and-Swap observes equal initial and final values, and hence succeeds. However, the value A was not maintained continuously, and the data could therefore be in an inconsistent state. The Compare-and-Swap should fail in this circumstance but it does not. - The failure mode is eliminated in practice by operating on a double-operand. The second operand is a counter that is incremented each time Compare-and-Swap is executed. A Compare-and-Swap can update erroneously only if both the original shared variable and the counter are the same as their original values obtained when the update process began. Since a counter may take many hours to increment to bring it to its original state, it is extremely unlikely to find both the counter and the shared variable unchanged when they have indeed changed. - The use of cache residence as an indicator of whether a value was maintained continuously or not was proposed by 2ù. A Compare-and-Swap must find a value in cache in order to succeed. Prior to a Compare-and-Swap, a Prepare Load instruction can bring an item into cache in preparation for a subsequent Compare-and-Swap. The detection of continuous residence has been implemented on the MIPS R-4000, DEC Alpha, and SPARC 9 processors, as mentioned earlier. - Uniprocessor multiprogrammed machines have to be protected as indicated in 2ù from incorrect behavior by detecting which of the active users issues a Prepare Load. The User ID is stored with the reservation. The user that issues a matching Compare-and-Swap must have a user ID that matches the one stored with the reservation. The enhancement eliminates the A-B-A problem for the Compare-and-Swap somewhat differently than the implementations now in use. It has the advantage that no special reservation registers are required, and all storage can be in the cache line, with minimal extra logic required for the instruction implementation. The implementation permits existing Compare-and-Swap codes to be carried over to the more protected environment. The enhancement is the following: 1. Each processor has a special new register called COUNTER. The value is arbitrarily

• set at power on, and is advanced and read as indicated in the subsequent steps. 2. The Prepare Load produces a double-word value, of which only the first word is obtained from memory. A Prepare Load that finds a cache miss, advances the COUNTER. This value together with the single-word operand of the instruction form a double-word pair that is loaded into a pair of general registers, just as if a double-word load had been executed. This double word is also stored into cache. The general registers are part of the context of the process, and are preserved during a context swap, and restored after the context swap, so that the value obtained from COUNTER persists through context swaps. (The word in memory corresponding to the second half of the double word must be reserved for this operation, even though it is not read from memory.) 3. A Prepare Load that finds a cache hit reads the double word from cache. This is the value of the shared operand and the value of the counter variable at the time the data were written to cache. 4. A Compare-and-Swap fails if it misses in cache. It reads the value of the first operand from memory, advances the value of the COUNTER, stores a double word in cache, and returns a double word to load into a register pair. 5. A Compare-and-Swap that hits in cache compares the double word there to the double word in the register pair. If they are equal, the Compare-and-Swap then issues an invalidate to all other caches to obtain Write ownership of the shared variable. When two or more processes issue the invalidate concurrently, exactly one succeeds in obtaining Write ownership of the variable. If the ownership is obtained successfully, the Compare-and-Swap succeeds. The Compare-and-Swap then writes a new value of the single-word shared variable to memory. (The word in memory corresponding to the next consecutive word must be reserved for this operation, even though it is not written.) If the comparison fails, the Compare-and-Swap then does the steps that occur when the Compare-and-Swap misses in cache. That is, the single-word operand is read from memory, COUNTER is advanced, the operand and the counter value are written to cache, and stored in a register pair in the processor. The implementation is nonblocking in the sense that if multiple processors attempt to Compare-and-Swap, one must succeed. This is crucial for supporting nonblocking algorithms. - The counter that differentiates one instance of a variable from others must have a number of different states great enough to cover the maximum number of simultaneous ownerships of a variable to distinct instances of that variable in time. - A major advantage of this idea over commercial implementations that now exist is that the reservation can survive a context swap. In the present commercial implementations mentioned, the reservation registers are cleared when a context swap occurs, in order to assure correct behavior of the program. The implementation described here places the reservation in a register visible to the program that is swapped out during a context swap, and thus permits multiple reservations to be outstanding simultaneously. This implementation tends to reduce thrashing that may be caused by context swaps at times when many different processes access shared variables and have reservations active. References 1ù H. S. Stone, High-Performance Computer Architecture, Reading, MA: Addison-Wesley, 1987. 2ù R. K. Treiber, "Systems programming: coping with parallelism," IBM Research Report RJ 5118, (April 1986).